USC Viterbi
School of Engineering

# SystemVerilogCSP: Modeling Digital Asynchronous Circuits Using SystemVerilog Interfaces

Arash Saifhashemi[1]
Peter A. Beerel[1,2]
[1]Ming Hsieh Dept. of Electrical Engineering, University of Southern California
[2]Fulcrum Microsystems, Calabasas CA, 91302.

CPA 2011
University of Limerick, Ireland

USC

# Outline

USC Viterbi School of Engineering

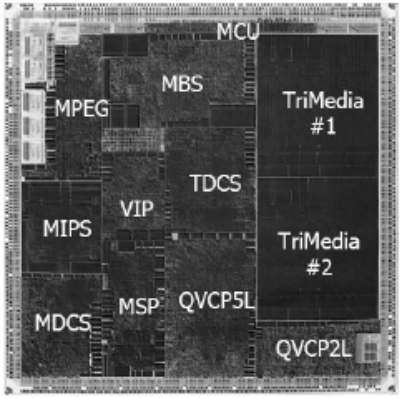
- Introduction
  - Why asynchronous circuit design?
  - Hardware description languages
- SystemVerilog Abstract Communication– Basic Features
  - Channels - Send, Receive
  - Channel status and mixed-level simulation
- SystemVerilog Abstract Communication– Extended Features
  - Peek and Probe
  - Split and synchronized communication
  - One-to-many and one-to-any channels
- Results and Conclusion

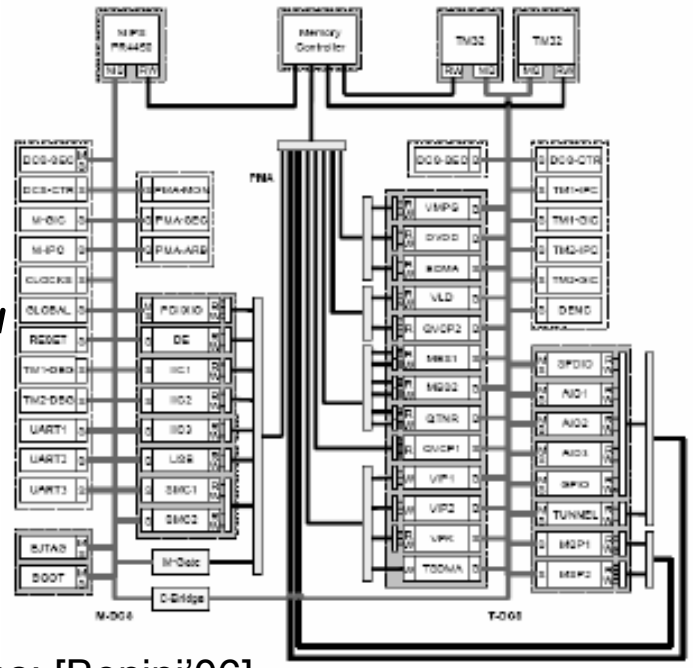
# Why Asynchronous Circuit Design?

USC Viterbi School of Engineering

- Systems on chip: chips are becoming distributed systems
  - Communication-dominated
  - No globally synchronous clock
- Asynchronous alternative
  - Local handshaking: CSP-like communication
- Benefits
  - Higher speed
    - Shorter globally critical paths
  - Lower power consumption
    - Remove power-hungry global clock
    - Modules active only when necessary
  - Robustness to variations
    - Process, voltage, and temperature

**60 IP blocks**
**350 RAMs:**
**Communication Bottleneck**



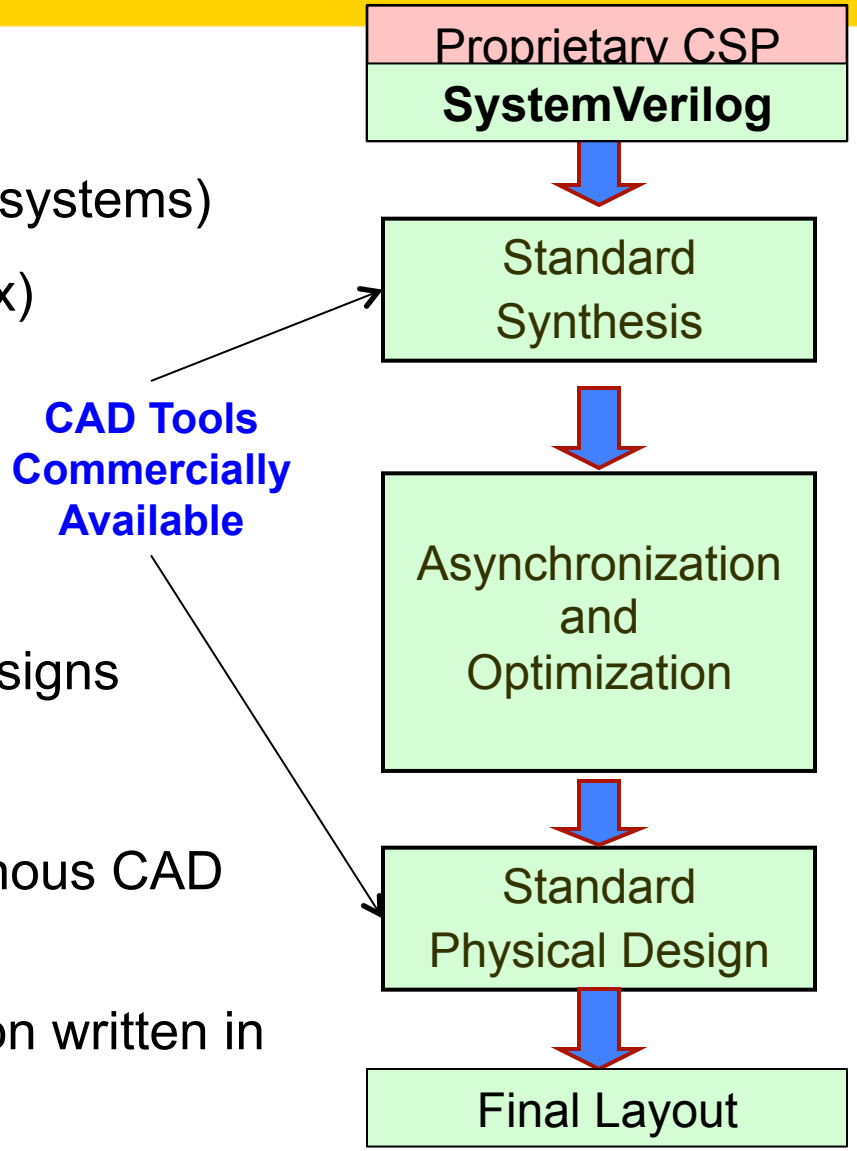



Pictures: [Benini'06]

# Asynchronous Circuit Design - Today

USC Viterbi School of Engineering

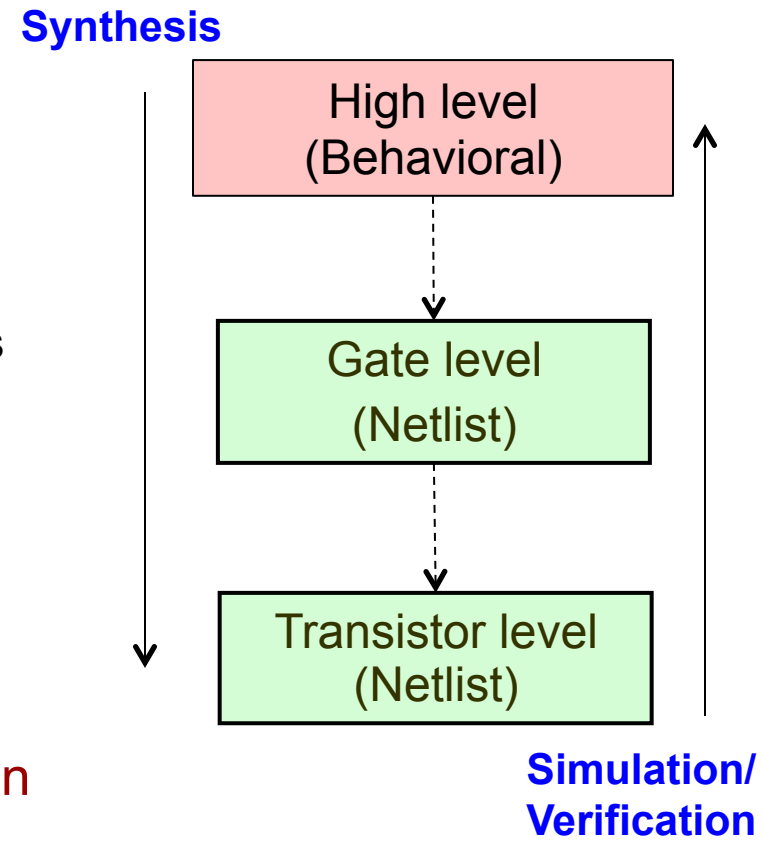
- Applications
  - Ethernet Switches (Fulcrum Microsystems)
  - Ultra high-speed FPGAs (Achronix)
  - Multi-core network on Chip
  - Ultra low-power chip design
- Basic challenges
  - Lack of CAD tools to automate designs
- Proteus design flow (USC)
  - Leverage off of available synchronous CAD tools
  - Starting at a high-level specification written in **SystemVerilogCSP**



# Hardware Description Languages

USC Viterbi School of Engineering

- Desirable features of an HDL
  - Concurrency e.g.: A=B || (C=D ; E=F)
  - Timing e.g.: A=B after 5ns
  - Support for various levels of abstraction
  - Support by commercial CAD tools
  - Support for both synchronous & asynchronous
- Communication abstraction
  - Ease of design
  - Design usability: protocols evolve and change
  - Architecture evaluation before implementation
  - Ease of adoption by synchronous designers
- CSP as a basis for a hardware description language
  - Suitable for modeling abstract communication
  - Lacks some desirable features



# Previous Work

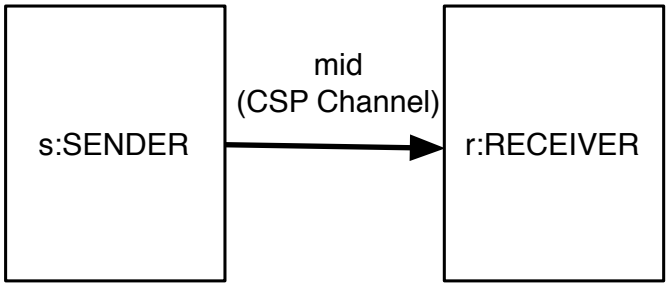
USC Viterbi School of Engineering

- **New Language inspired by CSP**
  - Have limited CAD tool support - LARD [Edwards et al], Tangram [Berkel et al], CHP [Martin]
- **Software languages**
  - No inherent support for timing, limited CAD tool support - JCSP [Welch et. al]
- **VHDL**
  - Fine grained concurrency is cumbersome [Frankild et al, Renaudin et al, Myers et al]
- **VerilogCSP**
  - Verilog Programming Language Interface: very slow; cannot handle multi-channel modules [Saifhashemi et al]
  - Verilog macros are cumbersome and do not support extensions
- **SystemVerilog (Superset of Verilog)**
  - Initial implementations promising but do not address extensions [Tiempo]

# CSP Communication Channels

USC Viterbi School of Engineering

- Abstract communication between processes




$$SENDER = (mid!v \rightarrow SENDER)$$
$$RECEIVER = (mid?x \rightarrow RECEIVER)$$

- No notion of **hardware implementation details**
- Semantics based on **events** on **channels** between independent **processes** [Hoare'04]

# Outline

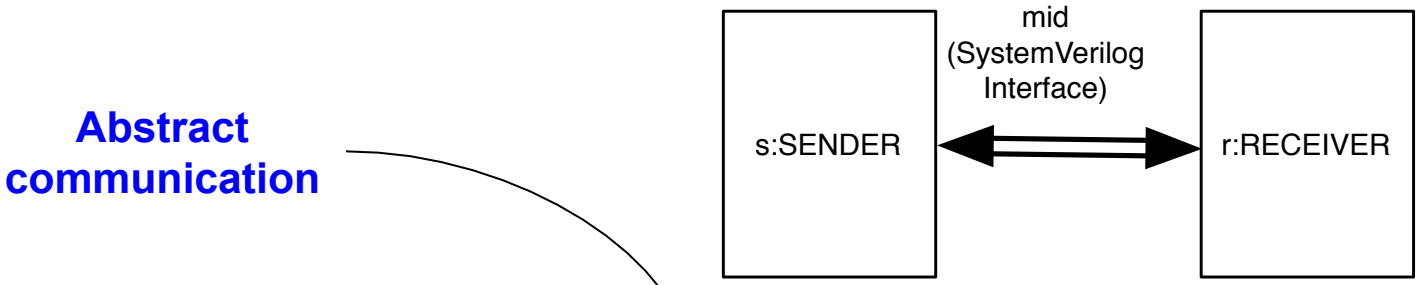
USC Viterbi School of Engineering

- Introduction
  - Why asynchronous circuit design?
  - Hardware description languages
- SystemVerilog Abstract Communication– Basic Features
  - Channels - Send, Receive
  - Channel status and mixed-level simulation
- SystemVerilog Abstract Communication- Advanced Features
  - Peek and Probe
  - Split and synchronized communication
  - One-to-many and one-to-any channels
- Results and Conclusion

# Abstract SystemVerilog Channels

USC Viterbi School of Engineering

- Our approach
  - Use SystemVerilog interface to abstract **channel wires** as well as **Send/Receive tasks**




```
module Sender (interface R);
  parameter WIDTH = 8;
  logic [WIDTH-1:0] v;
  always
  begin
  v={$random()}%2**(WIDTH-1);
  R.Send(v);
  #10ns;
  end
endmodule
```

```
module Receiver (interface L);
  parameter WIDTH = 8;
  logic [WIDTH-1:0] x;
  always
  begin
   L.Receive(x);
    #15ns;
  end
endmodule
```

# Behind The Scenes: Channel Interface

USC Viterbi School of Engineering

- Channel details encapsulated within an "Interface"
- Implementation details (**below**) hidden from user
  - Greatly simplifies **debugging** and **evaluation** of the design

```
typedef enum {idle, r_pend, s_pend}      ChannelStatus;
typedef enum {P2PhaseBD, P4PhaseBD}      ChannelProtocol;

interface Channel;
  parameter WIDTH = 8;
  parameter ChannelProtocol hsProtocol = P2PhaseBD;
  ChannelStatus status = idle;// Status of a channel
  logic req=0, ack=0;          // Handshaking signals
  logic hsPhase=1;             // Used in two-phase
                               // handshaking
  logic [WIDTH-1:0] data;      // Data being communicated
endinterface: Channel
```

# Interface Send and Receive Tasks

USC Viterbi School of Engineering

**Arbitrary handshaking protocol**
**Support most commonly used**

```
task Send (input logic
    [WIDTH-1:0] d);
  begin
    data = d;
    req = 1;
    status = s_pend;
    wait (ack == 1 );
    req = 0;
    wait (ack == 0 );
    status = idle;
  end
endtask
```

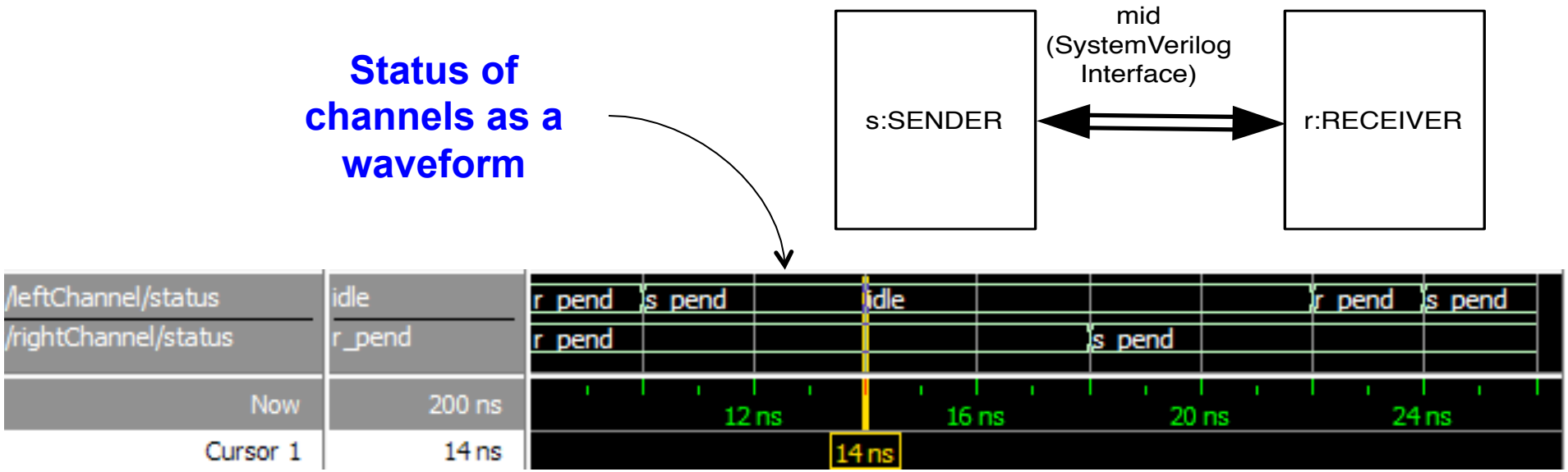
```
task Receive(output logic
  [WIDTH-1:0] d);
  begin
    status = r_pend;
    wait (req == 1 );
    d = data;
    ack = 1;
    wait (req == 0 );
    ack = 0;
    status = idle;
  end
endtask
```

- Send/Receive tasks are **analogous** to CSP's ! (output) and ? (input)
- Semantics are based on synchronization of concurrent **processes** using SystemVerilog's notion of **update** and **evaluation events**

# Viewing Channel Status

USC Viterbi School of Engineering

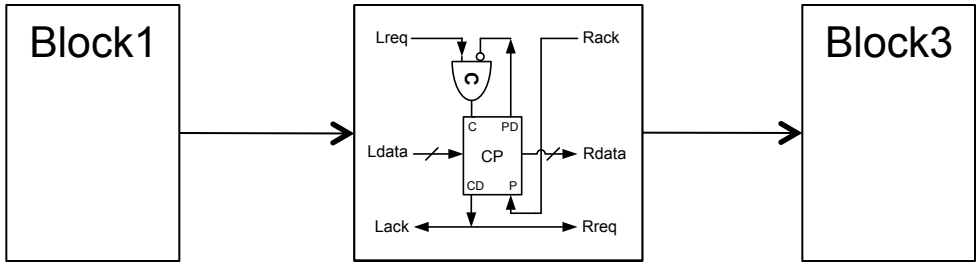
- Enumerated types make viewing channel status inherent to **all standard SystemVerilog simulators**
- The designer can **monitor** if and what processes are engaged in the communication over time



# Supports Mixed-Levels of Abstraction

USC Viterbi School of Engineering

- **Completed** blocks can be simulated with others **still at behavioral level**




```
module mp_fb_csp (interface L, interface R);
    logic data;
    always
    begin
        L.Receive(data);
        R.Send(data);
    end
endmodule
```

**High-level description of the buffer (Before synthesis)**

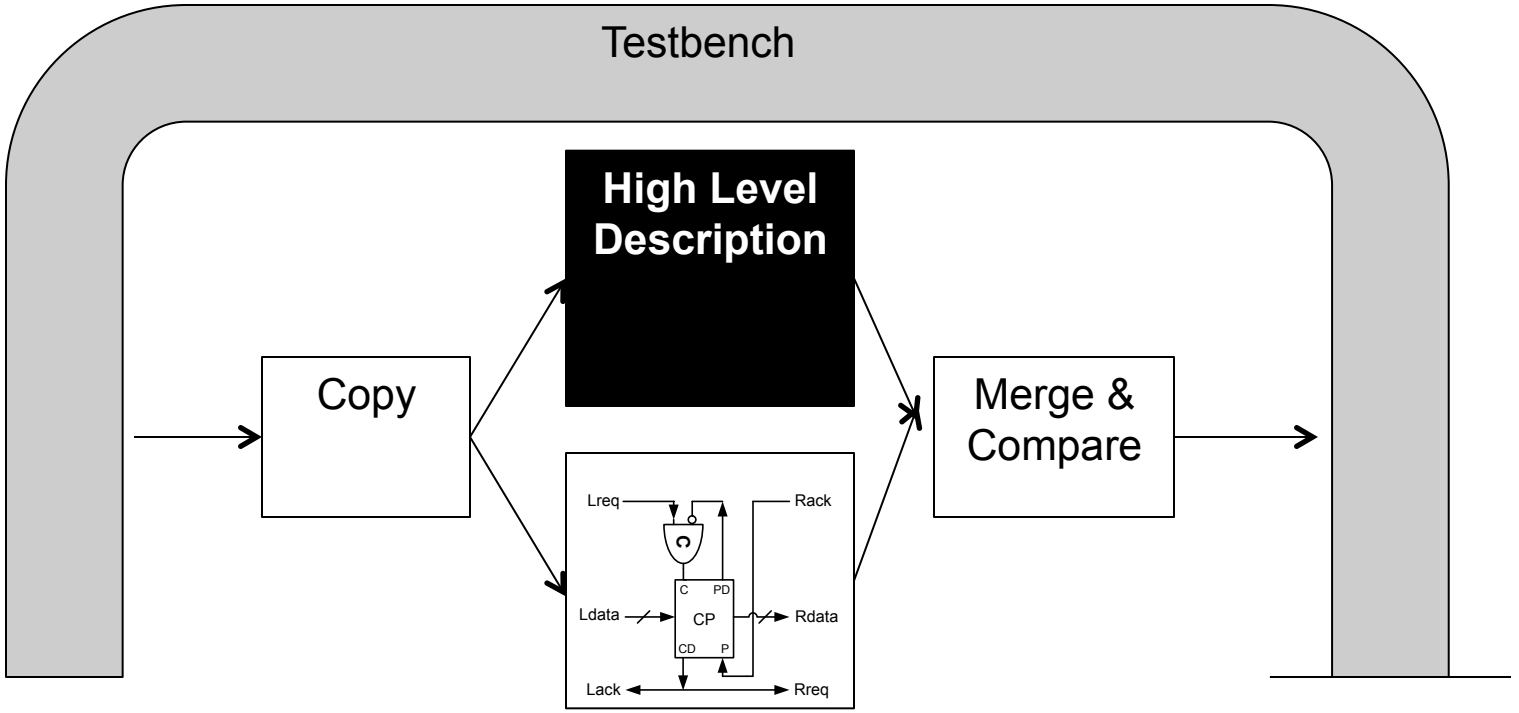
**Gate-level description of the buffer (After synthesis)**

```
module mp_fb_gate (interface L, interface R);
  celement     ce(L.req, pd_bar, c);
  not   inv    (pd_bar, pd);
  cap_pass     cp (c, L.ack, R.ack, pd, L.data, R.data);
endmodule
```

# Supports Design Verification

USC Viterbi School of Engineering

- Co-simulation: Implemented circuit vs. original circuit
- It is important to use the same Testbench
  - Sometimes very complicated
  - Verifies correct implementation
- No need for Shims [Saifhashemi'05]



# Outline

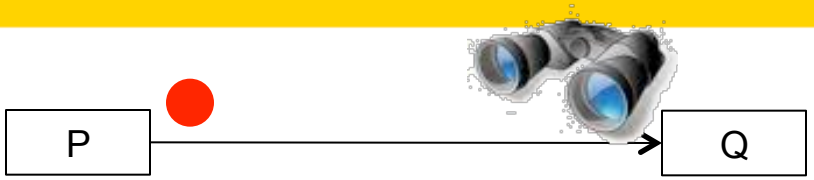
USC Viterbi School of Engineering

- Introduction
  - Why asynchronous circuit design?
  - Hardware description languages
- SystemVerilog Abstract Communication– Basic Features
  - Channels - Send, Receive
  - Channel status and mixed-level simulation
- SystemVerilog Abstract Communication- Advanced Features
  - Peek and Probe
  - Split and synchronized communication
  - One-to-many and one-to-any channels
- Results and Conclusion

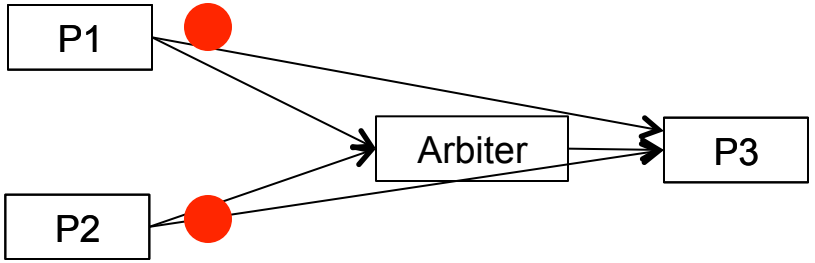
# Peek and Probe

USC Viterbi School of Engineering

- Peek
  - Sample data without committing to communication
- Probe
  - Is the channel idle?
  - Usually used for arbitration




```
task Peek (output logic[WIDTH-1:0] d);
  wait (status == s_pend );
  d = data;
endtask
```



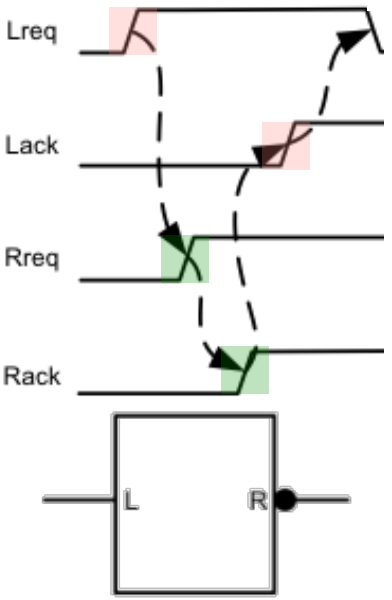

```
wait(ch0.status!=idle && ch1.status!= idle);
winner = Arbitrate (ch0.status, ch1.status);

if(winner == 0)
  ch0.Receive(d);
if(winner ==1)
  ch1.Receive(d);
```

# Split Communication

USC Viterbi School of Engineering

- Handshaking of different channels might be **interleaved in implementation**
- Modeling interleaved behavior **at high level** is important for **early system evaluation**



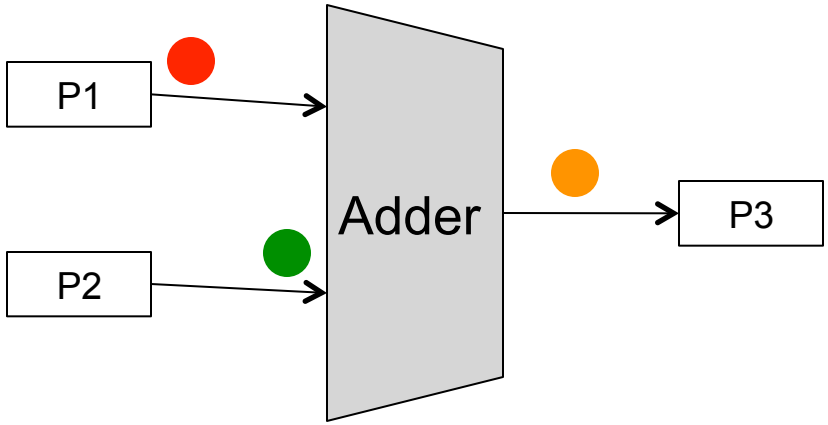

```
module buf (interface L, interface R);
    logic data;
    always
    begin
        L.Receive(data);
        R.Send(data);
    end
endmodule
```

```
module buf_split (interface L, interface R);
    logic data;
    always
    begin
        L.SplitReceive(data, 1);
        R.Send         (data);
        L.SplitReceive(data, 2);
    end
endmodule
```

# Synchronized Communications

USC Viterbi School of Engineering

- Sometimes implementation forces **correlation** of communication on **multiple channels**
  - Synchronized start
  - Synchronized finish
- Early performance evaluation of system requires modeling such behavior




**Concurrent body starts**

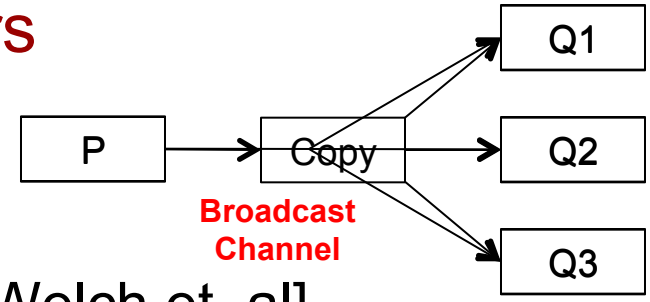
**Acts like a barrier**

```
always
 begin
  fork
     A.Receive(a, 1);
     B.Receive(b, 1);
  join
  fork
     A.Receive(a, 2);
     B.Receive(b, 2);
  join
  sum = a + b ;
  SUM.Send(sum);
 end
```

# One-To-Many Channels

USC Viterbi School of Engineering

- One sender to multiple receivers
  - Option 1: Use a copy block
    - Makes design **cumbersome**

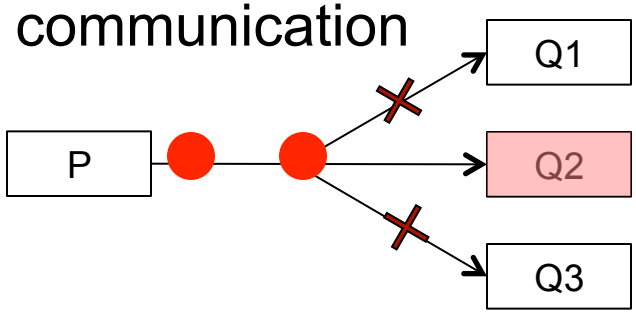



  - Option 2: Shared channels [JCSP, Welch et. al]
    - Sender and receiver send and receive **as if the channel is a normal one-to-one channel**
    - Top level module specifies the channel is broadcast
  - Shared channels are closer to hardware implementation
    - A shared data bus between sender and receivers
    - Separate *req* and *ack* signals for receiving processes.

# One-To-Any Channel

USC Viterbi School of Engineering

- One sender to multiple receiver - JCSP [Welch et. al]
  - Only one of the receiver participates in communication




```
task Receive(output logic[WIDTH-1:0] d);
  status = r_pend;
  wait (req == hsPhase );
  if (ONE2ANY)
    req = 'z; // Inhibits
              //other receivers
  d = data;
  ack = hsPhase;
  status = idle;
endtask
```

```
always
  begin : main
    wait (L.status != idle);
    randValue = {$random()} % 3 ;
    if (randValue ==1)
      L.Receive(x);
    else
      begin
        #0;
        disable main;
      end
  end
```

# Outline

USC Viterbi School of Engineering

- Introduction
  - Why asynchronous circuit design?
  - Hardware description languages
  - CSP Communication Channels
- SystemVerilog Abstract Communication– Basic Features
  - Channels - Send, Receive
  - Channel status and mixed-level simulation
- SystemVerilog Abstract Communication- Advanced Features
  - Peek and Probe
  - Split and synchronized communication
  - One-to-many and one-to-any channels
- Results and Conclusion

# Results – Simulation Run-Times

USC Viterbi School of Engineering

- Comparison to VerilogCSP [Saifhashemi'05]
  - Simulation time of a linear pipeline with depth of 10
  - Platform: Sun UltraSPARC, Modelsim SE 6.6 simulator
  - 12%-20% improvement

| Number of data items | 100K | 200K | 300K | 400K | 500K |
|---|---|---|---|---|---|
| Simulation time in Seconds (VerilogCSP) | 45.14 | 76.38 | 107.60 | 139.57 | 170.62 |
| Simulation time in Seconds (SystemVerilogCSP) | 40.12 | 65 | 89.70 | 115.52 | 141.99 |
| Ratio | 1.12 | 1.17 | 1.19 | 1.20 | 1.20 |

# Conclusions

USC Viterbi School of Engineering

- CSP-like communication and extensions can be modeled using SystemVerilog interfaces
- Features and advantages
  - Ease of design: abstract communication, channel status
  - Mixed asynchronous and synchronous designs can be modeled in same language and simulation environment
  - Extensions: more accurate modeling of implemented hardware
  - Make adoption of asynchronous technology easier
- Currently being used to teach the course EE-552 Asynchronous VLSI at the University of Southern California
- Future work
  - Automated synthesis from SystemVerilogCSP

# Supports Design Verification

USC Viterbi School of Engineering

- Testing DUT:
  - Initially, modeled in SystemVerilogCSP
  - Later implemented in gates
- It is important that Testbench does change
  - Sometimes very complicated
  - Communicates with other blocks
  - Verifies correct implementation
- No need for Shims [Saifhashemi'05]

**Testbench should stay the same as DUT is implemented**